SPM project: Parallel Prefix

Gaspare Ferraro, 520549

Master Degree in Computer Science - University of Pisa

ferraro@gaspa.re

November 1, 2018

1 Introduction

In this report we will analyze, theoretically and practically, the resolution of the problem of the (parallel) prefix sum:

Given a vector $x = \langle x_0, x_1, \ldots, x_{n-1} \rangle$ *and a binary operation* $\oplus$ *compute the vector* $y = \langle x_0, x_0 \oplus x_1, x_0 \oplus x_1 \oplus x_2, \ldots, x_0 \oplus x_1 \oplus \ldots \oplus x_{n-1} \rangle$.

In literature this operation is also called (inclusive) scan or partial sum.

For the analysis of the problem we have to make two assumptions:

- The binary operation $\oplus$ is associative ($a \oplus (b \oplus c) = (a \oplus b) \oplus c$) and commutative ($a \oplus b = b \oplus a$), this is an important assumption as we will see in the next chapters the order of the operations may not be preserved.
- The size of the input vector is a power of 2, not a strong assumption, as all the algorithms we will present could be easily generalize to all the sizes, but it only helps to simplify some operations.

2 Sequential algorithm

The sequential algorithm simple compute each element of the vector y as follow:

$$y_i = \begin{cases} x_0 & \text{for } i = 0 \\ x_i \oplus y_{i-1}, & \text{for } 0 < i < n \end{cases}$$

This algorithm is optimal in a sequential model as it has a running time of $\mathcal{O}(n)$, assuming that $\oplus$ is $\mathcal{O}(1)$, and performs $n-1$ calls to $\oplus$ operation.

3 Parallel architecture design

The problem of computing the prefix sum vector is a classical example of a problem that have an optimal solution in a sequential model but that can be optimized in a parallel model. The optimization is not in terms of total complexity or in the number of $\oplus$ operations performed (which are already optimals in the sequential algorithm) but in terms of completion time. When the number of available workers is more than one we can trade-off more total work for less completion time.

We will now introduce two different algorithms that solve in an efficient way the prefix sum problem in a parallel model.

3.1 Block-based algorithm

The idea behind the first parallel algorithm is to compute the prefix vector in three phases:

- Partitionate the input vector in blocks and compute in parallel the prefix vector of each of them.
- In the second phase we put the final element of each block in a temporary vector and compute its prefix vector.
- Finally for each block i (except for the first one) add in parallel the $(i-1)$th element of the temporary vector.

At the end of the three phases the final vector contains the correct prefixes.

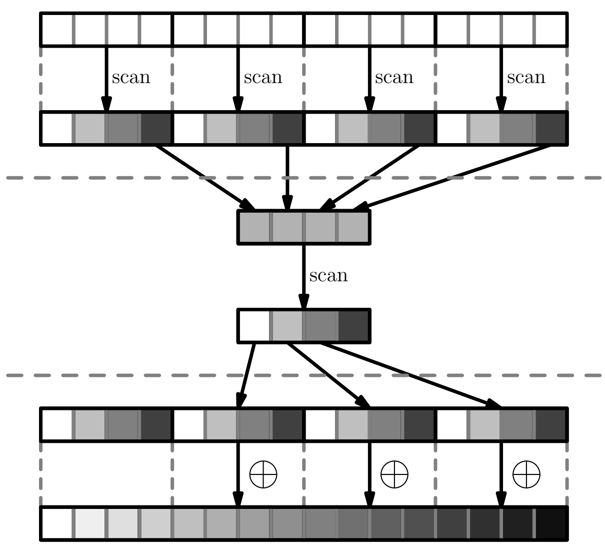


Figure 1: Block-based algorithm graphic representation

3.2 Circuit-based algorithm

Another class of algorithms are the one based on a circuit-like representation. A generic prefix circuit C is a collection of tasks $T_1, T_2, \ldots, T_t,$. A task T_i is a set of operations (l, r) that means to compute $x_r = x_r \oplus x_l$. The tasks must be computed sequentially from 1 to t but the advantage is that all the operations in a same task can be performed in parallel without race conditions.

For example the sequential algorithm seen before can be represented as a prefix circuit $S(n) = \{T_1, T_2, \ldots, T_{n-1}\}$ where $T_i = \{(i-1, i)\}$.

We present now the simple (but still efficient) prefix circuit $P(2^m)$:

$$P(2^m) = \{T_1, T_2, \ldots, T_m, T_{m+1}, \ldots, T_{2m-1}\}$$

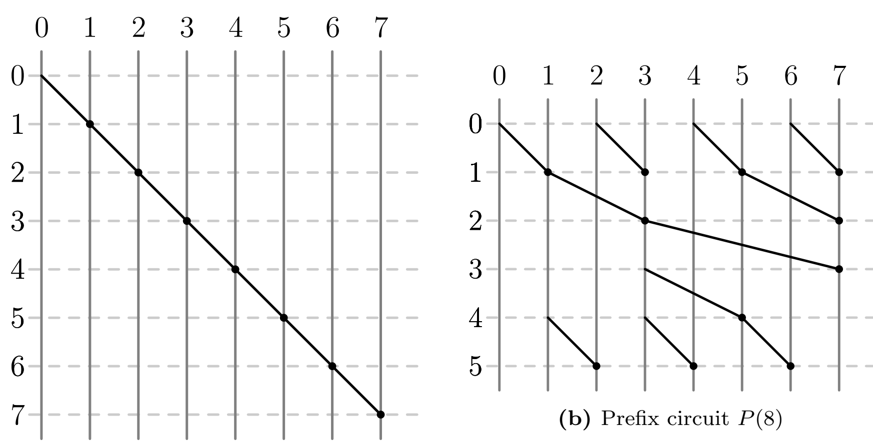
where for $i = 1, \ldots, m$:

$$T_i = \{(2k^i - 2^{i-1} - 1, 2k^i - 1) \mid k = 1, \ldots, 2^{m-i}\}$$

and for $i = 1, \ldots, m-1$:

$$T_{m+i} = \{(k2^{m-i} - 1, k2^{m-i} + 2^{m-i-1} - 1) \mid k = 1, \ldots, 2^i - 1\}$$

In the first phase (from 1 to m) we compute operations like in a reduction tree, in the second phase (from $m+1$ to $2m-1$) we compute the final value of each prefix as the sum of only two others values.



(a) Prefix circuit $S(8)$

(b) Prefix circuit $P(8)$

Figure 2: Examples of prefix circuits

As we can see in these simple examples, for $n = 8$ the sequential algorithm needs 7 units of time against the only 5 of the parallel algorithm (obviously if the number of workers is at least 4).

4 Performance modeling

For the theoretical performance modeling we will denote with:

- $n = 2^m$ the size of the input vector ($m = \log_2 n$).
- p the parallelism degree (for simplicity $p \leq n$).
- $t_\oplus$ is the time to compute a single $\oplus$ operation.

4.1 Sequential algorithm

The completion time of the sequential algorithm is simply $T_C^s = (n-1) \times t_\oplus$.

4.2 Block-based algorithm

The parallel completion time of the block-based algorithm is the sum of the completion times of all of the three phases:

$$T_{C_p}^b \geq T_{1_p} + T_{2_p} + T_{3_p} \tag{1}$$

Where:

- $T_{1_p} = (n/p) \times t_\oplus$ is the time to compute the prefix sums of a single block (as all the p blocks are performed in parallel).
- $T_{2_p} = (p) \times t_\oplus$ is the time to compute the prefix sums over the temporary vector.
- $T_{3_p} = (n/p) \times t_\oplus$ (or 0 if $p = 1$) is the time to add a value to a single block (except the first one).

So the total completion time is $T_{C_p}^b \geq (2n/p + p) \times t_\oplus$.

In this analysis we assume that n is a multiple of p and all the blocks have the same size, if this is not the case we can consider the block size as the size of the biggest block.

4.3 Circuit-based algorithm

In the prefix circuit $P(n)$ we have a total of $2m-1$ tasks that must be executed sequentially. As said before all the operations in the same task can be performed in parallel at the same time, so a generic task T_i has a completion time of $t_\oplus$.

So the completion time of the circuit based algorithm is $T_{C_p}^c \geq 2\log_2(n) - 1$.

However this analysis works only if the parallelism degree is high enough to allow to performs all the operations in T_i at the same time ($p \geq n/2$).

When this is not the case a task T_i is perfomed in $\max(1, |T_i|/p) \times t_\oplus$ time and it can be demonstrated that $T_{C_p}^c \geq 2(\log_2 p + n/p) \times t_\oplus$.

p	T_C^s	$T_{C_p}^b$	$T_{C_p}^c$
1	$(n-1) \times t_\oplus$	$n \times t_\oplus$	$2n \times t_\oplus$
2	$(n-1) \times t_\oplus$	$n \times t_\oplus$	$(n+2) \times t_\oplus$
4	$(n-1) \times t_\oplus$	$(n/2+4) \times t_\oplus$	$(n/4+2) \times t_\oplus$
8	$(n-1) \times t_\oplus$	$(n/4+8) \times t_\oplus$	$\times t_\oplus$
...	...	...	...
n/2	$(n-1) \times t_\oplus$	$(n/2+1) \times t_\oplus$	$\times t_\oplus$
n	$(n-1) \times t_\oplus$	$(n+2) \times t_\oplus$	$\times t_\oplus$

Table 1: Comparison of the three algorithms with different parallelism degree

4.4 Comparison table

What we can see from the table is that

5 Implementations structure and details

All the implementations are written in $C++17$, the source code is available as attachment with the report or on GitHub
(https://github.com/GaspareG/ParallelPrefix).

5.1 Sequential algorithm

The sequential implementations

5.2 Block-based algorithm

TODO

5.3 Circuit-based algorithm

TODO

6 Experimental validation

6.1 experiments details

6.2 benchmark results

7 Conclusion